

Exceptions

(<http://docs.oracle.com/javase/tutorial/essential/exceptions/index.html>)

Objectives

- Exception Handling
 - try block
 - catch block
 - finally block
 - custom exception class

Exceptions

- **Exception**: Error beyond the control of a program. When an exception occurs, the program will terminate abruptly.
- When a program is executing something occurs that is not quite normal from the point of view of the goal at hand.
- For example:
 - a user might type an invalid filename;
 - An accessed file does not exist or might contain corrupted data;
 - a network link could fail;
 - ...
- Circumstances of this type are called *exception conditions* in Java and are represented using objects (All exceptions descend from the `java.lang.Throwable`).

Exceptions

- The following program causes an exception.

```
ExceptionDemo_1.java *
public class ExceptionDemo_1 {
    public static void main (String[] args)
    {
        int x=5, y=0;
        System.out.println(x/y);
        System.out.println("Hello");
    }
}
```

Exceptions are pre-defined data (Exception classes) thrown by JVM and they can be caught by code in the program

```
Output - Chapter04 (run)
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at ExceptionDemo_1.main(ExceptionDemo_1.java:4)
Java Result: 1
BUILD SUCCESSFUL (total time: 2 seconds)
```

Kinds of Exceptions

- `java.lang.Throwable` (implements `java.io.Serializable`)
 - `java.lang.Error`
 - `java.lang.Exception`
 - `java.lang.RuntimeException`

Checked Exceptions
(We may not use the try catch blocks)

Unchecked- Exceptions
Program Bugs
(We must use the try catch blocks or throw)

Refer to the Java.lang documentation for more information.

```
1 public class ExceptionDemo_1 {
2     public static void main (String[] args)
3     { int[] a= { 1,2,3,4,5};
4       int n=10;
5       for (int i=0;i<n;i++)
6           System.out.print(" " + a[i] + ",");
7     }
8 }
9
```

Output - Chapter04 (run)

```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
1,2,3,4,5, at ExceptionDemo_1.main(ExceptionDemo_1.java:6)
Java Result: 1
BUILD SUCCESSFUL (total time: 1 second)
```

```
1 public class ExceptionDemo_1 {
2     public static void main (String[] args)
3     { int[] a= { 1,2,3,4,5};
4       int n=10;
5       try
6       { for (int i=0;i<n;i++)
7           System.out.print(" " + a[i] + ",");
8       }
9       catch(Exception e) // general exception
10      { System.out.println(e);
11      }
12 }
```

Output - Chapter04 (run)

```
run:
1,2,3,4,5,java.lang.ArrayIndexOutOfBoundsException: 5
BUILD SUCCESSFUL (total time: 0 seconds)
```

Two Kinds of Exception

a) Checked Exception

Các lớp extends từ lớp Throwable ngoại trừ RuntimeException và Error được gọi là checked exception.

ví dụ như Exception, SQLException vv. Các checked exception được kiểm tra tại compile-time.

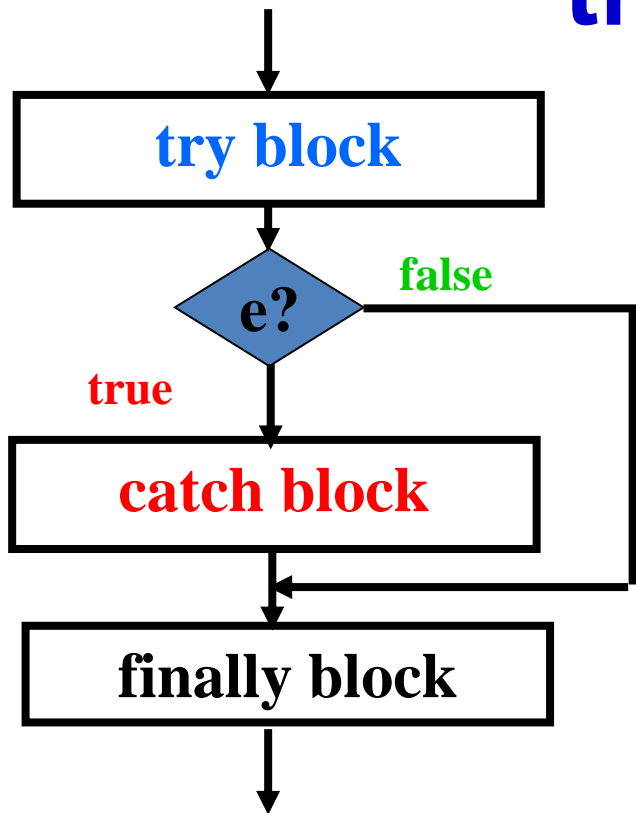
b) Unchecked Exception

Các lớp extends từ RuntimeException được gọi là unchecked exception

ví dụ: ArithmeticException, NullPointerException,

ArrayIndexOutOfBoundsException,... Các ngoại lệ unchecked không được kiểm tra tại compile-time mà chúng được kiểm tra tại runtime.

Catching exceptions: try catch finally



If no exception is thrown in the try block, all catch blocks are bypassed

If an exception arises, the first matching catch block, if any, is executed, and the others are skipped

```

try {
    < statements may cause exceptions >
}

catch ( ExceptionType1 e1 ) {
    < statements handle the situation 1>
}

catch ( ExceptionType2 e2) {
    < statements handle the situation 2>
}

finally {
    < statements are always executed >
}
    
```

Catching specific/general-level exception

```
ExceptionDemo_1.java x
1 public class ExceptionDemo_1 {
2     public static void main (String[] args)
3     { int x=6, y=0;
4         try
5         { System.out.println(x/y);
6             // other statements
7         }
8         catch( ArithmeticException e)
9         { System.out.println(e);
10             y=2;
11         }
12         finally
13         { System.out.println("Hello");
14             System.out.println(x/y);
15         }
16     }
17 }
```

Output - Chapter04 (run)

```
run:
java.lang.ArithmeticException: / by zero
Hello
3
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
ExceptionDemo_1.java * x
1 public class ExceptionDemo_1 {
2     public static void main (String[] args)
3     { int x=6, y=0;
4         try
5         { System.out.println(x/y);
6             // other statements
7         }
8         catch(Exception e) // general exception
9         { e.printStackTrace();
10             y=2;
11         }
12         finally
13         { System.out.println("Hello");
14             System.out.println(x/y);
15         }
16     }
17 }
```

Type conformity: father=son;

Output - Chapter04 (run)

```
run:
Hello
java.lang.ArithmeticException: / by zero
3
    at ExceptionDemo_1.main(ExceptionDemo_1.java:5)
BUILD SUCCESSFUL (total time: 0 seconds)
```


Throwing exceptions in methods

May we intentionally throw an exception? → YES

```

1 public class ExceptionDemo_1 {
2     public int divide1(int a, int b) throws
3         ArithmeticException
4     { return a/b;
5     }
6     public int divide2(int a, int b)
7     { if (b==0) throw new ArithmeticException
8         ("Hey. Denominator:0");
9         return a/b;
10    }
11    public static void main (String[] args)
12    { ExceptionDemo_1 obj= new ExceptionDemo_1();
13        try
14        { System.out.println(obj.divide1(6,0));
15        }
16        catch(Exception e) // general exception
17        { System.out.println(e);
18        }
19    }
20 }

```

Output - Chapter04 (run)

```

run:
java.lang.ArithmeticException: / by zero
BUILD SUCCESSFUL (total time: 0 seconds)

```

```

1 public class ExceptionDemo_1 {
2     public int divide1(int a, int b) throws
3         ArithmeticException
4     { return a/b;
5     }
6     public int divide2(int a, int b)
7     { if (b==0) throw new ArithmeticException
8         ("Hey. Denominator:0");
9         return a/b;
10    }
11    public static void main (String[] args)
12    { ExceptionDemo_1 obj= new ExceptionDemo_1();
13        try
14        { System.out.println(obj.divide2(6,0));
15        }
16        catch(Exception e) // general exception
17        { System.out.println(e);
18        }
19    }
20 }

```

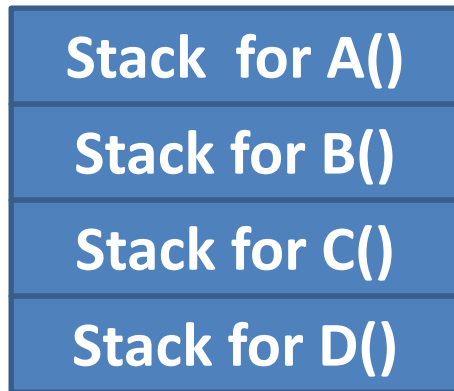
Output - Chapter04 (run)

```

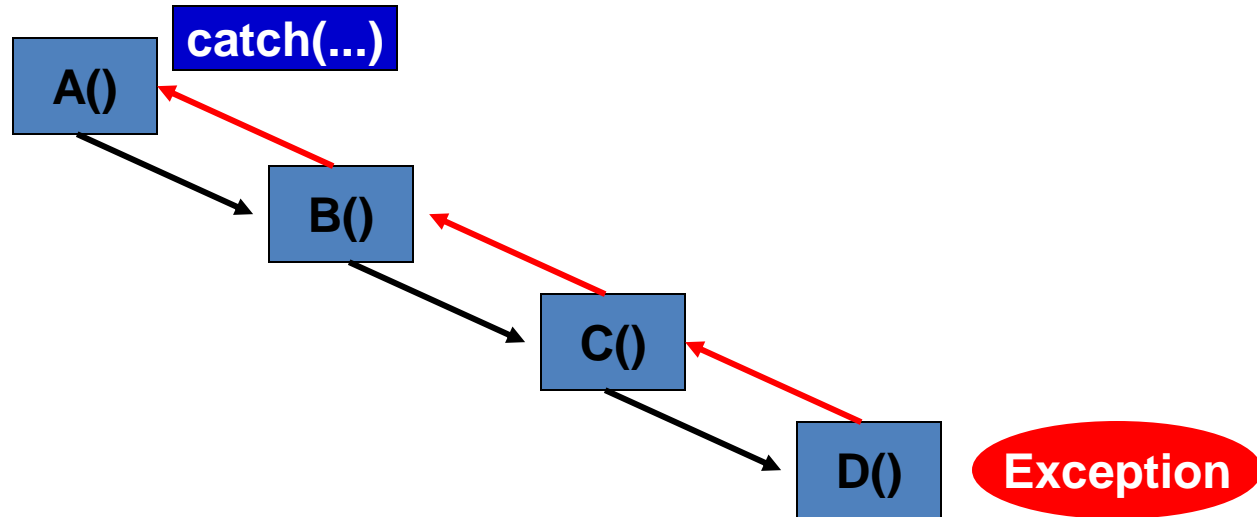
run:
java.lang.ArithmeticException: Hey. Denominator:0
BUILD SUCCESSFUL (total time: 0 seconds)

```

Exception Propagations



Stack trace



When an exception occurs at a method, program stack is containing running methods (method A calls method B,...). So, we can trace statements related to this exception.

Exception Propagations

```
1 public class ExceptionPropagate {
2     public void mA()
3     {
4         mB();
5     }
6     public void mB()
7     {
8         mC();
9     }
10    public void mC()
11    {
12        System.out.println(5/0);
13    }
14    public static void main(String[] args) {
15        ExceptionPropagate obj= new ExceptionPropagate();
16        obj.mA();
17    }
18 }
```

Output - FirstPrj (run) x

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at ExceptionPropagate.mC(ExceptionPropagate.java:12)
    at ExceptionPropagate.mB(ExceptionPropagate.java:8)
    at ExceptionPropagate.mA(ExceptionPropagate.java:4)
    at ExceptionPropagate.main(ExceptionPropagate.java:16)
```

Java Result: 1

Catching Exceptions...

Using try...catch to input an integer $10 \leq n \leq 50$

```
Scanner in = new Scanner(System.in);
boolean cont = true;
int n;
do {
    try {
        System.out.print("Enter a whole number: ");
        n = Integer.parseInt(in.nextLine());
        cont = false;
    } catch (Exception e) {
        System.out.println("Required integer!");
    }
} while (cont == true || n < 10 || n > 50);
```

The *finally* block (1)

- A try block may optionally have a finally block associated with it.
- The code within a finally block is *guaranteed* to execute no matter what happens in the try/catch code that precedes it.
 - The try block executes to completion without throwing any exceptions whatsoever.
 - The try block throws an exception that is handled by one of the catch blocks.
 - The try block throws an exception that is ***not*** handled by ***any*** of the catch blocks

Nesting of try/catch Blocks

- A try statement may be nested inside either the try or catch block of another try statement.








```
try {
    // Pseudo code.
    open a user-specified file
}
catch (FileNotFoundException e) {
    try {
        // Pseudo code.
        open a DEFAULT file instead ...
    }
    catch (FileNotFoundException e2) {
        // Pseudo code.
        attempt to recover ...
    }
}
```

Nesting of try/catch Blocks

```

1
2  package session07;
3
4  public class Nest_test {
5      public static void main(String args[]){
6          int [] a={1,2,3,4,5}; //Parent try block
7          try{
8              //Child try block1
9              System.out.println("Value of a[3] : " + a[3]);
10
11             try{
12                 System.out.println("Inside block1");
13                 int b =a[4]/0;
14                 System.out.println("Value of b : " + b);
15             }
16             catch(ArithmeticException e1){
17                 System.out.println("Arithmetic Exception");
18             }
19
20             System.out.println("Value of a[5] : " + a[5]);
21         }
22
23         catch(ArrayIndexOutOfBoundsException e4){
24             System.out.println("ArrayIndexOutOfBoundsException");
25             System.out.println("Inside parent try catch block");
26         }
27
28         System.out.println("Next statement..");
29     }
30 }

```

Notifications	Output - Java_Basic (run) X
	run:
	Value of a[3] : 4
	Inside block1
	Arithmetic Exception
	ArrayIndexOutOfBoundsException
	Inside parent try catch block
	Next statement..
	BUILD SUCCESSFUL (total time: 0 seconds)

Creating Your Own Exception Classes (1)

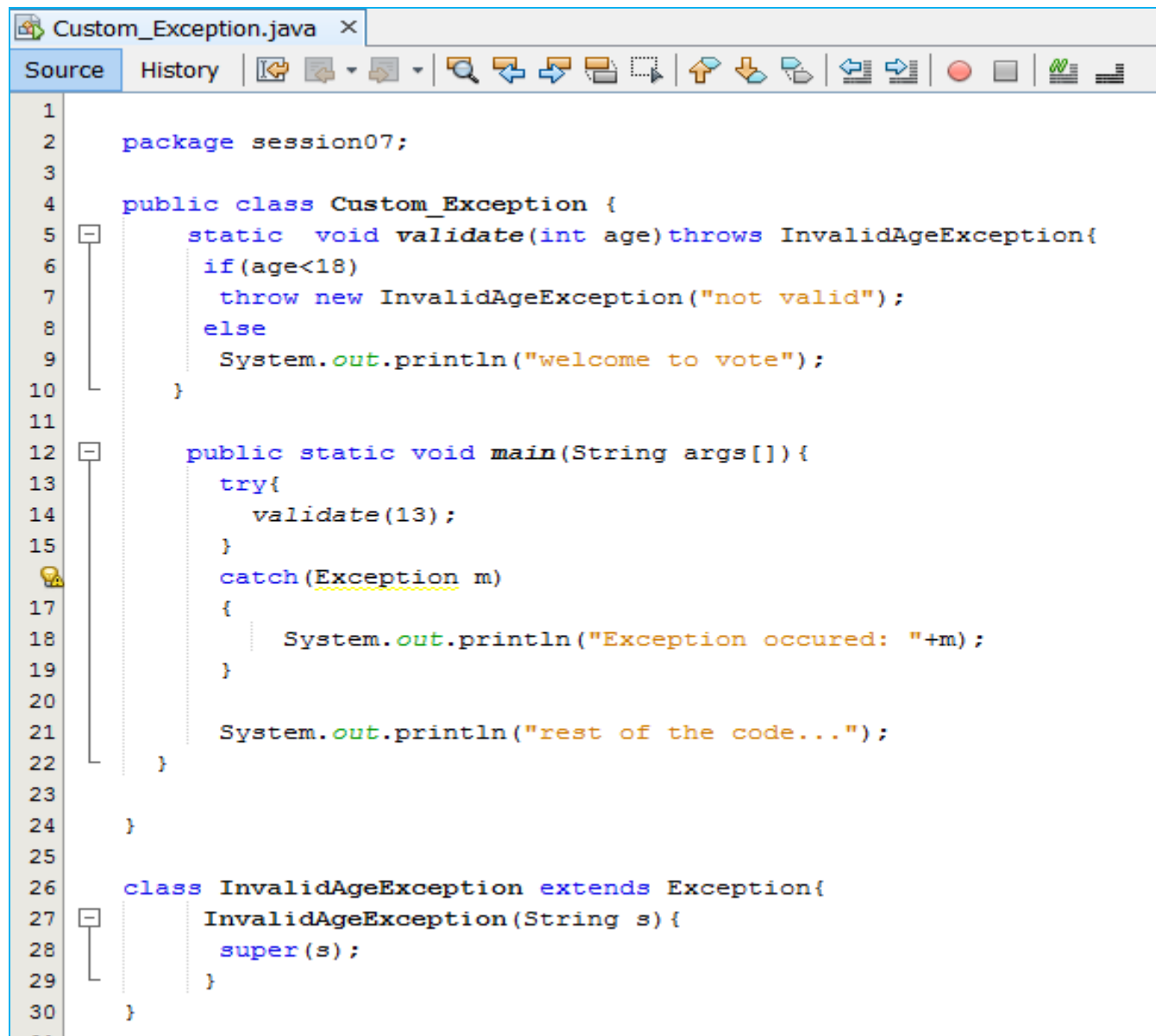
- Decide whether you want a checked or a runtime exception.
 - Checked exceptions should extend `java.lang.Exception` or one of its subclasses.
 - Runtime exceptions should extend `java.lang.RuntimeException` or one of its subclasses

Creating Your Own Exception Classes (2)

Create your own exception class with it's constructor

```
class InvalidAge extends Exception{  
    public InvalidAge(String mes) {  
        super(mes);  
    }  
}
```

Creating Your Own Exception Classes (3)



```

1
2  package session07;
3
4  public class Custom_Exception {
5      static void validate(int age) throws InvalidAgeException{
6          if(age<18)
7              throw new InvalidAgeException("not valid");
8          else
9              System.out.println("welcome to vote");
10     }
11
12     public static void main(String args[]){
13         try{
14             validate(13);
15         }
16         catch(Exception m)
17         {
18             System.out.println("Exception occurred: "+m);
19         }
20
21         System.out.println("rest of the code...");
22     }
23
24 }
25
26 class InvalidAgeException extends Exception{
27     InvalidAgeException(String s){
28         super(s);
29     }
30 }

```

Creating Your Own Exception Classes (4)

Output - Java_Basic (run)



run:



Exception occurred: session07.InvalidAgeException: not valid
rest of the code...



BUILD SUCCESSFUL (total time: 0 seconds)

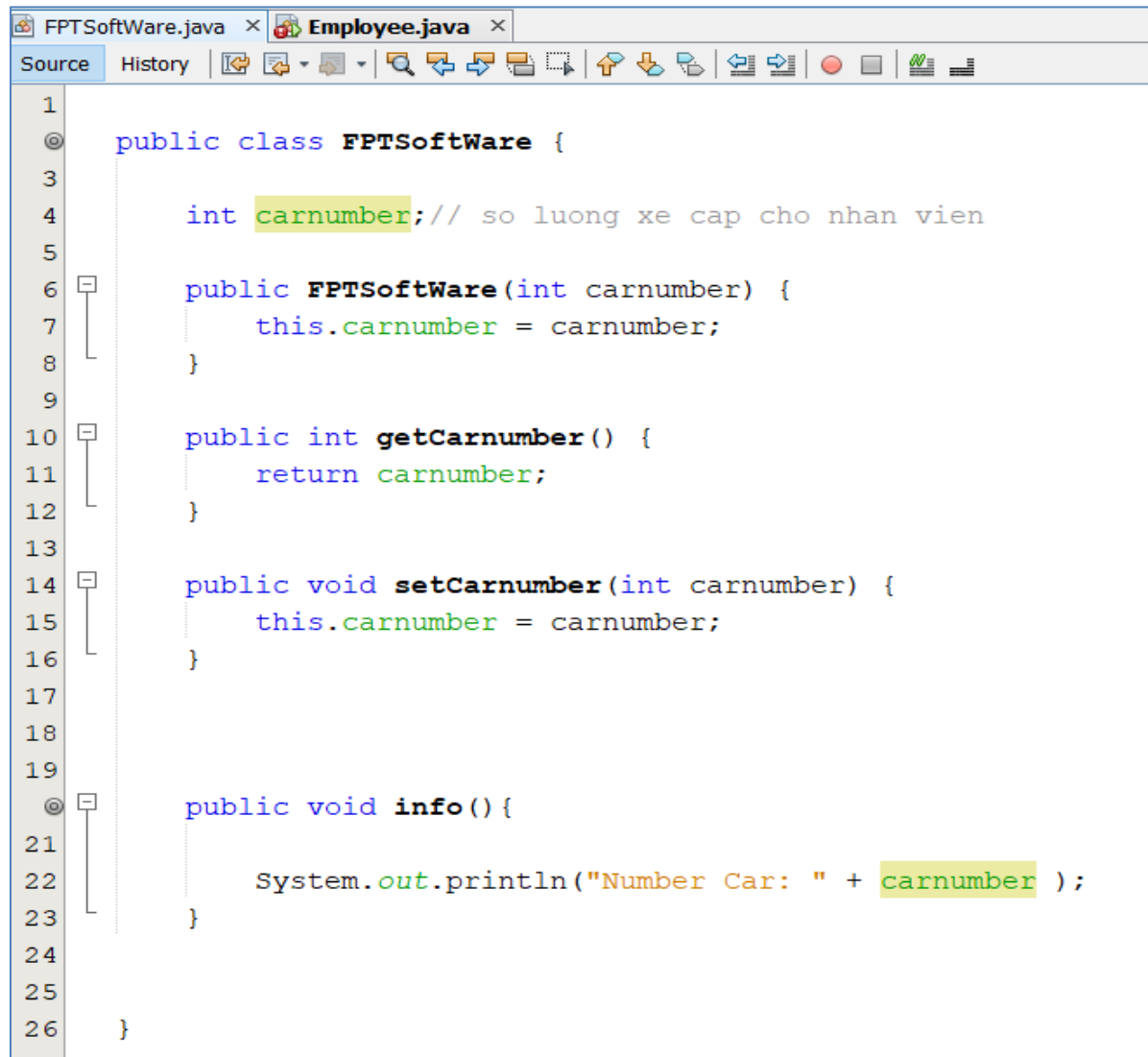


|

Exceptions and Overriding

- When you extend a class and override a method, the Java compiler insists (đòi hỏi) that all exception classes thrown by the new method must be the same as, or subclasses of, the exception classes thrown by the original method.

Exceptions and Overriding



```

1  public class FPTSoftware {
2
3      int carnumber; // so luong xe cap cho nhan vien
4
5      public FPTSoftware(int carnumber) {
6          this.carnumber = carnumber;
7      }
8
9      public int getCarNumber() {
10         return carnumber;
11     }
12
13     public void setCarNumber(int carnumber) {
14         this.carnumber = carnumber;
15     }
16
17     public void info() {
18         System.out.println("Number Car: " + carnumber );
19     }
20 }
  
```

Exceptions and Overriding

```

1
2 public class Employee extends FPTSoftWare {
3
4     static int n=0; // so luong nhan vien
5
6     public Employee(int carnumber) {
7         super(carnumber);
8     }
9
10
11     @Override
12     public void info() throws Exception { // vi Exception la lop cha cua cac lop ngoai le
13
14         System.out.println("Result : " + (carnumber/n));
15     }
16
17
18     public static void main(String[] args) {
19
20         Employee e1=new Employee(10);
21         try {
22             e1.info();
23         } catch (Exception ex) {
24             System.out.println(ex);
25         }
26     }
27
28 }

```

Exceptions and Overriding

```

1
2 public class Employee extends FPTSoftWare {
3
4     static int n=0; // so luong nhan vien
5
6     public Employee(int carnumber) {
7         super(carnumber);
8     }
9
10
11     @Override
12     public void info() throws ArithmeticException { // ArithmeticException la lop con nen khong bi loi
13
14         System.out.println("Result : " + (carnumber/n));
15     }
16
17
18     public static void main(String[] args) {
19
20         Employee e1=new Employee(10);
21         try {
22             e1.info();
23         } catch (Exception ex) {
24             System.out.println(ex);
25         }
26     }
27
28 }

```

Output - JavaApplication25 (run)

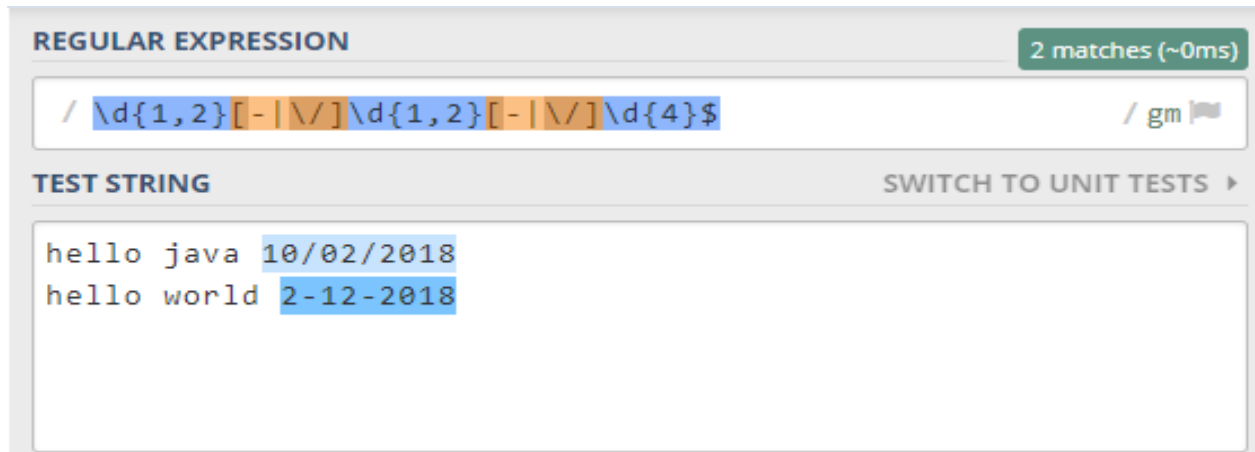
```

run:
java.lang.ArithmeticException: / by zero
BUILD SUCCESSFUL (total time: 0 seconds)

```

Java Regex

Java Regex hoặc Regular Expression (biểu thức chính quy) là một API để định nghĩa một mẫu để tìm kiếm hoặc thao tác với chuỗi. Nó được sử dụng rộng rãi để xác định ràng buộc trên các chuỗi như xác thực mật khẩu, email, kiểu dữ liệu datetime, ...



Java Regex

Lớp Pattern

No.	Phương thức	Mô tả
1	<code>static Pattern compile(String regex)</code>	biên dịch regex đã cho và trả về thể hiện của Pattern.
2	<code>Matcher matcher(CharSequence input)</code>	tạo một matcher khớp với đầu vào đã cho với mẫu.
3	<code>static boolean matches(String regex, CharSequence input)</code>	Nó biên dịch biểu thức chính quy và tìm kiếm các chuỗi con từ chuỗi input phù hợp với mẫu regex.
4	<code>String[] split(CharSequence input)</code>	chia chuỗi input đã cho thành mảng các kết quả trùng khớp với mẫu đã cho.
5	<code>String pattern()</code>	trả về mẫu regex.

Java Regex

Ví dụ sử dụng Regex trong Java - tìm kiếm chuỗi con

Ví dụ sau tìm tất cả các chuỗi ngày tháng có định dạng dd-mm-yyyy hoặc dd/mm/yyyy trong chuỗi văn bản text1 và xác minh xem chuỗi text2 và text3 có định dạng ngày tháng hay không.

Định nghĩa regex: `\d{1,2}[-/] \d{1,2}[-/] \d{4}`

`\d{1,2}`: nghĩa là một số có 1 hoặc 2 chữ số (ngày và tháng).

`[-/]`: nghĩa là ký tự - hoặc /.

`\d{4}`: nghĩa là một số có 4 chữ số (năm).

Java Regex

```

DemoRegex.java x
Source History
1
2 package demo05;
3
4 import java.util.regex.Matcher;
5 import java.util.regex.Pattern;
6
7 public class DemoRegex {
8     public static void main(String[] args) {
9         String text1 = "Hello java regex 2-12-2018, hello world 12/12/2018";
10        Pattern pattern = Pattern.compile("\\d{1,2}[-|/]\\d{1,2}[-|/]\\d{4}");
11        Matcher matcher = pattern.matcher(text1);
12
13        System.out.println("Ngày tháng trong chuỗi text1: " + text1);
14        while (matcher.find()) {
15            System.out.println(text1.substring(matcher.start(), matcher.end()));
16        }
17
18        String text2 = "2/12/2018";
19        String text3 = "12/12/aaaa";
20        pattern = Pattern.compile("^\\d{1,2}[-|/]\\d{1,2}[-|/]\\d{4}$");
21        System.out.println("\nChuỗi " + text2 + " có định dạng ngày tháng: "
22            + pattern.matcher(text2).matches());
23
24        System.out.println("Chuỗi " + text3 + " có định dạng ngày tháng: "
25            + pattern.matcher(text3).matches());
26    }
27 }

```

Java Regex

```
Output - Week0200P (run) x
run:
Ngày tháng trong chuỗi text1: Hello java regex 2-12-2018, hello world 12/12/2018
2-12-2018
12/12/2018

Chuỗi 2/12/2018 có định dạng ngày tháng: true
Chuỗi 12/12/aaaa có định dạng ngày tháng: false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java Regex

Các lớp ký tự Regex

Regex	Mô tả
[...]	trả về một ký tự phù hợp
[abc]	a, b, hoặc c
[^abc]	Bất kỳ ký tự nào ngoại trừ a, b, hoặc c
[a-zA-Z]	a tới z hoặc A tới Z
[a-d[m-p]]	a tới d, hoặc m tới p: [a-dm-p]
[a-z&&[def]]	d, e, hoặc f
[a-z&&[^bc]]	a tới z, ngoại trừ b và c: [ad-z]
[a-z&&[^m-p]]	a tới z, ngoại trừ m tới p: [a-lq-z]
[0-9]	0 tới 9

Java Regex

Regex	Mô tả
.	Bất kỳ ký tự nào
^	Có 2 cách sử dụng. 1. Đánh dấu bắt đầu của một dòng, một chuỗi. 2. Nếu sử dụng trong dấu [...] thì nó có nghĩa là phủ định.
\$	Đánh dấu Kết thúc của một dòng
\d	Bất kỳ chữ số nào, viết tắt của [0-9]
\D	Bất kỳ ký tự nào không phải chữ số, viết tắt của [^0-9]
\s	Bất kỳ ký tự trống nào (như dấu cách, tab, xuống dòng, ...), viết tắt của [\t\n\x0B\f\r]
\S	Bất kỳ ký tự trống nào không phải ký tự trống, viết tắt của [^\s]
\w	Bất kỳ ký tự chữ nào (chữ cái và chữ số), viết tắt của [a-zA-Z_0-9]
\W	Bất kỳ ký tự nào không phải chữ cái và chữ số, viết tắt của [^\w]
\b	Ranh giới của một từ
\B	Không phải ranh giới của một từ

Java Regex

Sử dụng String.matches(string_regex)

```

1
2  package demo05;
3
4  import java.util.Scanner;
5  /*
6   Ví dụ: Username có độ dài từ 6 đến 12 ký tự, không có khoảng trắng và không dấu:
7   */
8  public class DemoRegex02 {
9      public static void main(String[] args) {
10         Scanner sc=new Scanner(System.in);
11         String username;
12         System.out.print("Enter User Name:");
13         username=sc.nextLine();
14
15         if(username.matches("[a-z0-9_]{6,12}$")){
16             System.out.println("Valid");
17         }
18         else{
19             System.out.println("Invalid");
20         }
21     }
22 }

```

Java Regex

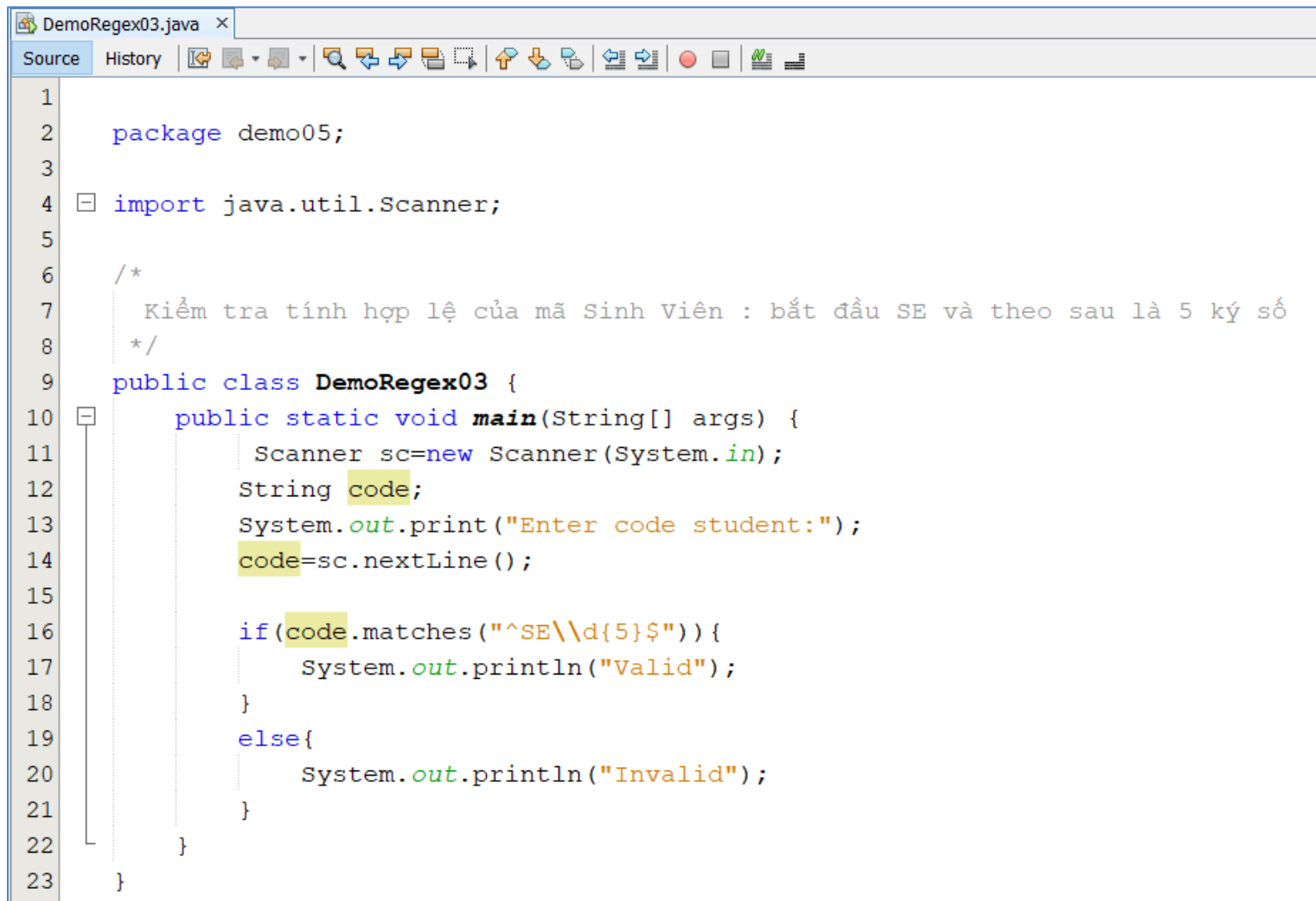
Sử dụng String.matches(string_regex)

```
Output - Week02OOP (run) x DemoRegex02.java x
run:
Enter User Name:Nguyen van An @$
Invalid
BUILD SUCCESSFUL (total time: 16 seconds)
```

```
Output - Week02OOP (run) x DemoRegex02.java x
run:
Enter User Name:thanh-phong
Valid
BUILD SUCCESSFUL (total time: 8 seconds)
```


Java Regex

Trong Java Regex bạn muốn nó hiểu các ký tự đó theo cách thông thường bạn cần thêm dấu \ ở phía trước.



```
1
2 package demo05;
3
4 import java.util.Scanner;
5
6 /*
7  Kiểm tra tính hợp lệ của mã Sinh Viên : bắt đầu SE và theo sau là 5 ký số
8  */
9 public class DemoRegex03 {
10     public static void main(String[] args) {
11         Scanner sc=new Scanner(System.in);
12         String code;
13         System.out.print("Enter code student:");
14         code=sc.nextLine();
15
16         if(code.matches("^SE\\d{5}$")){
17             System.out.println("Valid");
18         }
19         else{
20             System.out.println("Invalid");
21         }
22     }
23 }
```

Java Regex

```
Output - Week02OOP (run) x
run:
Enter code student:ab12345
Invalid
BUILD SUCCESSFUL (total time: 19 seconds)
```

```
Output - Week02OOP (run) x
run:
Enter code student:SE17001
Valid
BUILD SUCCESSFUL (total time: 15 seconds)
```

Summary

- Exception Handling
- Multiple Handlers
- Code Finalization and Cleaning Up (finally block)